CREATING A GRAPHIC USER INTERFACE (GUI) WITH PYTHON

Posted on February 24, 2018

Table of Contents

There are quite a number of **GUI packages** within Python and to be honest, I haven't tried out them all yet so I am not in a position to give any advice. For the purpose of this tutorial, I used **Tkinter** because... well, there's no specific reason. I did some searching and it looked possible with Tkinter so I just went ahead and tried it. If any of you have other suggestions, please feel free to share it in the comments section.

For our feedback control system, there will be a need to experiment with a few input values to get some measurement done. While it is possible to recompile and upload a new code with new values, it is not really the most effective way of doing things. Especially when there are many values that you want to experiment with. I wrote this piece of code so that the experiments can be carried out in a simpler and more effective manner. Whether for this project or for your own personal use, I hope that this post will be of some help to you guys 🙂

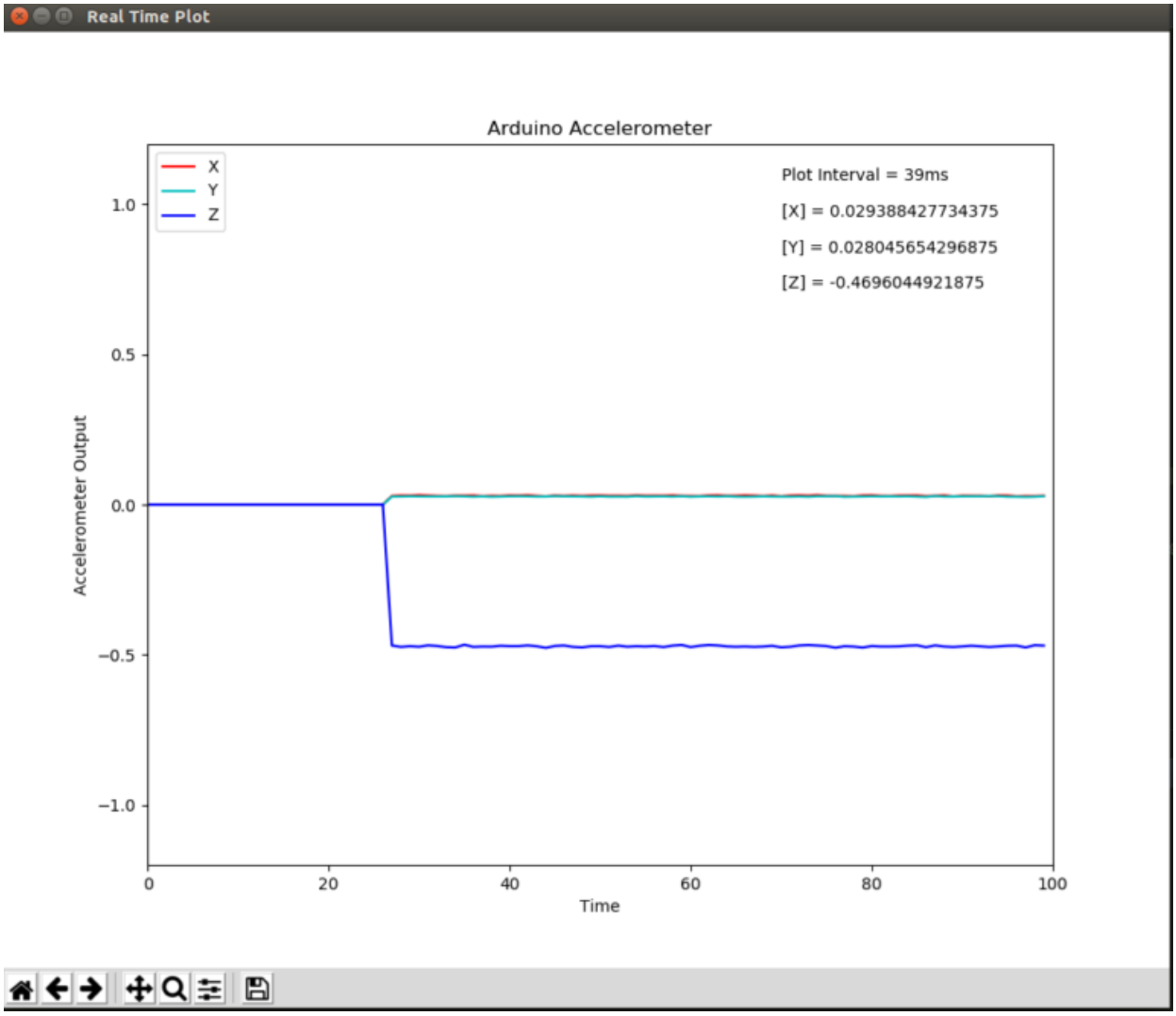
In order to follow this post, you will need to read up the previous section on **plotting a real time graph** first because we will be using the code from there.

Creating a Graphic User Interface (GUI) with Python

For this section, we will add on to the code in the previous chapter for **drawing multiple plots in a single graph**. We will place our graph from before unto the Tkinter's interface and add buttons and text boxes to create a GUI in a single window. In order to embed everything into Tkinter's window, there are some templates that we need to follow. I created a class to handle the GUI and the barebone template looks something like this:

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2TkAgg
import tkinter as Tk
from tkinter.ttk import Frame

class Window(Frame):
    def __init__(self, figure, master, SerialReference):
        Frame.__init__(self, master)
        self.entries = []
        self.setPoint = None
        self.master = master        # a reference to the master window
        self.serialReference = SerialReference      # keep a reference to our serial con
        self.initWindow(figure)     # initialize the window with our settings

    def initWindow(self, figure):
        self.master.title("Real Time Plot")
        canvas = FigureCanvasTkAgg(figure, master=self.master)
        toolbar = NavigationToolbar2TkAgg(canvas, self.master)
        canvas.get_tk_widget().pack(side=Tk.LEFT, fill=Tk.BOTH, expand=1)

def main():
    s = serialPlot(portName, baudRate, maxPlotLength, dataNumBytes, numPlots)   # initia
    s.readSerialStart() # starts background thread

    # plotting starts below
    fig = plt.figure()

    # put our plot onto Tkinter's GUI
    root = Tk.Tk()
    app = Window(fig, root, s)

    for i in range(numPlots):
        lines.append(ax.plot([], [])
    anim = animation.FuncAnimation(fig, s.getSerialData)

    root.mainloop()   # use this instead of plt.show() since we are encapsulating everyt
    s.close()
```

I have removed a lot of lines from the code above to simply stuffs so it will not run if you simply just copy and paste it (here's the **full Python code**). In order to put all the stuff we had into Tkinter, we have to pass a reference of our old plot's window to it. This is done through the sub class "Window". When we instantiate the subclass such as shown above, it will take our figure and place it within Tkinter's canvas (window). The result when you run the code now is this:



TKINTER WINDOW

You see that the window looks different from the original matplotlib window! We have now successfully embedded our plot into Tkinter's window. The next step is to create widgets such as buttons and text box to make it into a real GUI. For the purpose of this tutorial, I will make buttons and text boxes to scale the value from our accelerometer.

We can use the GUI to alter variables in our python code to do the scaling but that's not really fun is it? In this tutorial, we are going to send messages to our dear Arduino to ask it to do the scaling for us, and then send us the scaled values. Now, that's more interesting because we now have a bi-directional communication (take note that this does not happen concurrently) with the Arduino. While the Arduino is sending us values to plot, we can send the Arduino commands to tell it how to scale the values! Let's start from the Arduino code for this.

Arduino Code

```
// Global Variables
String g_recvString = "";
double g_scaleFactor = 1.0;

// other existing functions not shown
//...
//...
//...

void loop() {
  timeSync(loopTime);

  // Accelerometer Data
  sensor.getAccel(&accelData);
  double x = accelData.X / pow(2, 15) * g_scaleFactor;
  double y = accelData.Y / pow(2, 15) * g_scaleFactor;
  double z = accelData.Z / pow(2, 15) * g_scaleFactor;
  sendToPC(&x, &y, &z);
  getSerialData();
}

void getSerialData()
{
  while (Serial.available())
  {
    char input = Serial.read();
    g_recvString += input;
    if (input == '%')   // this is the end of message marker
    {
      int index = g_recvString.indexOf('%');
      String tmp = g_recvString.substring(0, index);  // remove the '%' sign
      g_scaleFactor = tmp.toFloat();
      g_recvString = "";
      break;
    }
  }
}
```

I find that the easiest (not necessarily the best) way to read ***float*** values from the Arduino is to read everything as strings, then convert it to ***float*** using the existing `toFloat()` method from the String class. We could of course ask python to send the bytes that makes up the ***float*** and then recombine them back in the Arduino. This would be exactly the same as how we send the data from Arduino to Python and it would be much more efficient and faster. However, we don't particularly need the high speed transfer since we are only sending commands intermittently, probably once in a blue moon with respect to how fast Arduino is sending data to Python. As such, let us go with the easy way out 🙂

Arduino provides us with `Serial.read()` API which reads a single byte from the Serial buffer. Since we are now sending via text (array of characters), we have to read multiple bytes before the number can be completed. For example, 10.24582 would require 8 reads since there are 8 characters and each character is a single byte. I added in a terminator character for our message and it is used to notify Arduino that it knows it can update the `g_scaleFactor` parameter. There are many other ways to achieve the same result so feel free to use your own logic.

Once the Arduino receives the '%' termination character, it will update the global variable `g_scaleFactor` which then scales the datas value by the specified amount. Here's the **full Arduino code** for those who need it.

Python Code

For this tutorial, we are going to create 1 label, 1 text box, and 1 button. By clicking the button, the program will send a command to Arduino to scale the values from the sensor by the value stated in the text box. In order to do this, we have to modify our Tkinter `Window` class as follows:

```
class Window(Frame):
    def __init__(self, figure, master, SerialReference):
        Frame.__init__(self, master)
        self.entry = None
        self.setPoint = None
        self.master = master        # a reference to the master window
        self.serialReference = SerialReference      # keep a reference to our serial con
        self.initWindow(figure)     # initialize the window with our settings

    def initWindow(self, figure):
        self.master.title("Real Time Plot")
        canvas = FigureCanvasTkAgg(figure, master=self.master)
        toolbar = NavigationToolbar2TkAgg(canvas, self.master)
        canvas.get_tk_widget().pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)

        # create out widgets in the master frame
        lbl1 = Tk.Label(self.master, text="Scaling Factor")
        lbl1.pack(padx=5, pady=5)
        self.entry = Tk.Entry(self.master)
        self.entry.insert(0, '1.0')     # (index, string)
        self.entry.pack(padx=5)
        SendButton = Tk.Button(self.master, text='Send', command=self.sendFactorToMCU)
        SendButton.pack(padx=5)

    def sendFactorToMCU(self):
        self.serialReference.sendSerialData(self.entry.get() + '%')     # '%' is our end
```

And we also have to add another method to our `serialPlot` class in order to send data to the Arduino.

```
def sendSerialData(self, data):
    self.serialConnection.write(data.encode('utf-8'))
```

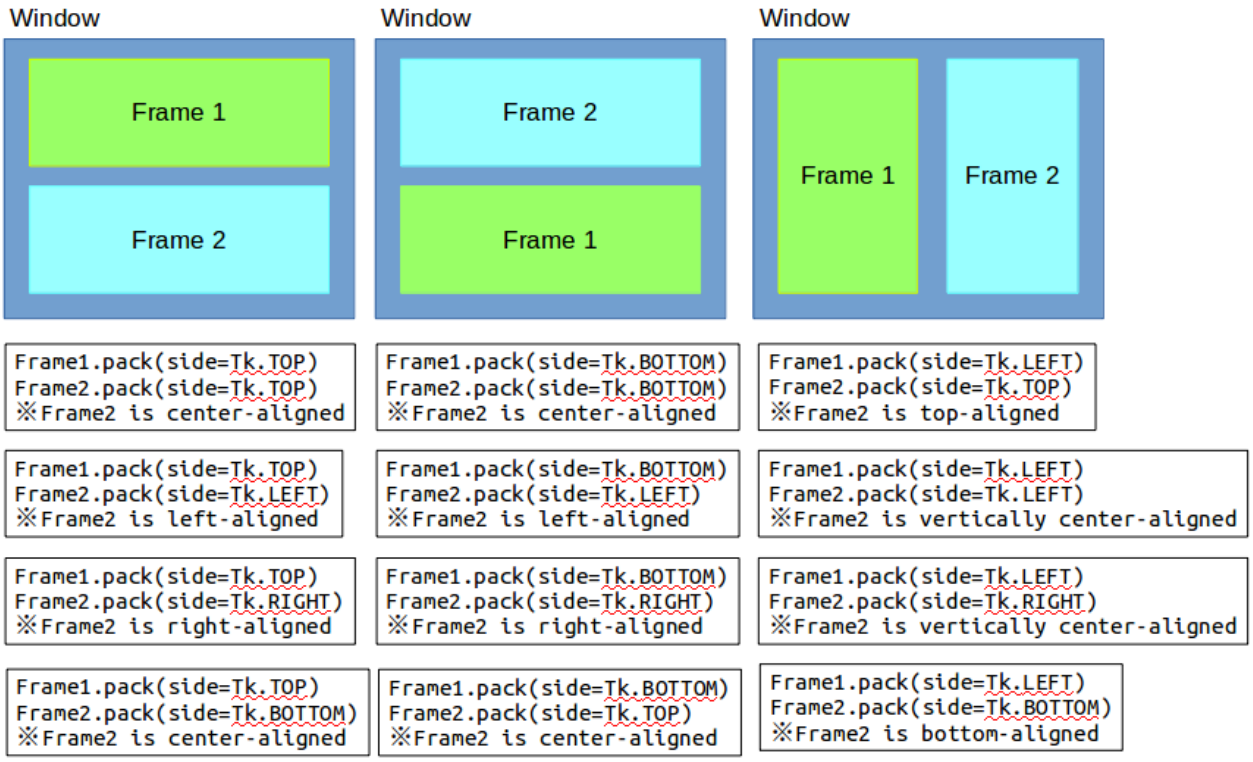
That is it. We're done (: Here's the **full Python code** for those who need it. Below is a video showing how the end product looks like.

Understanding object placement in Tkinter

The object placement in Tkinter might seem complicated at first but once you get used to it, it is not all that difficult. There are 3 main types of object placement method, namely “pack”, “grid” and “place”. I find that only “pack” seems to work when we embed our graph into the window so I’ll only talk about using “pack” for the purpose of this post. It is definitely possible mix the 3 different methods when you use an isolated “Frame” (I’ll get to this “Frame” in a moment) but since I did not need such an intricate GUI, I simply went with using “pack” alone. For those of you who are interested in the different placement methods, **here** is a good article about it.

We will dive straight into using “Frames” because it gives us the much needed versatility in placing our GUI objects. A frame is basically a window where you place all your objects. The thing about frames is that you can put a frame within a frame to divide the space within a window. In this manner, you can place your objects within a specific area.

There are 4 directions in which you can place your frames (and objects) using pack, namely TOP, BOTTOM, LEFT and RIGHT. If you take a look at the **documentation**, when nothing is specified, Tkinter uses TOP as a default for pack. Take a look at the image below for how the placement works.

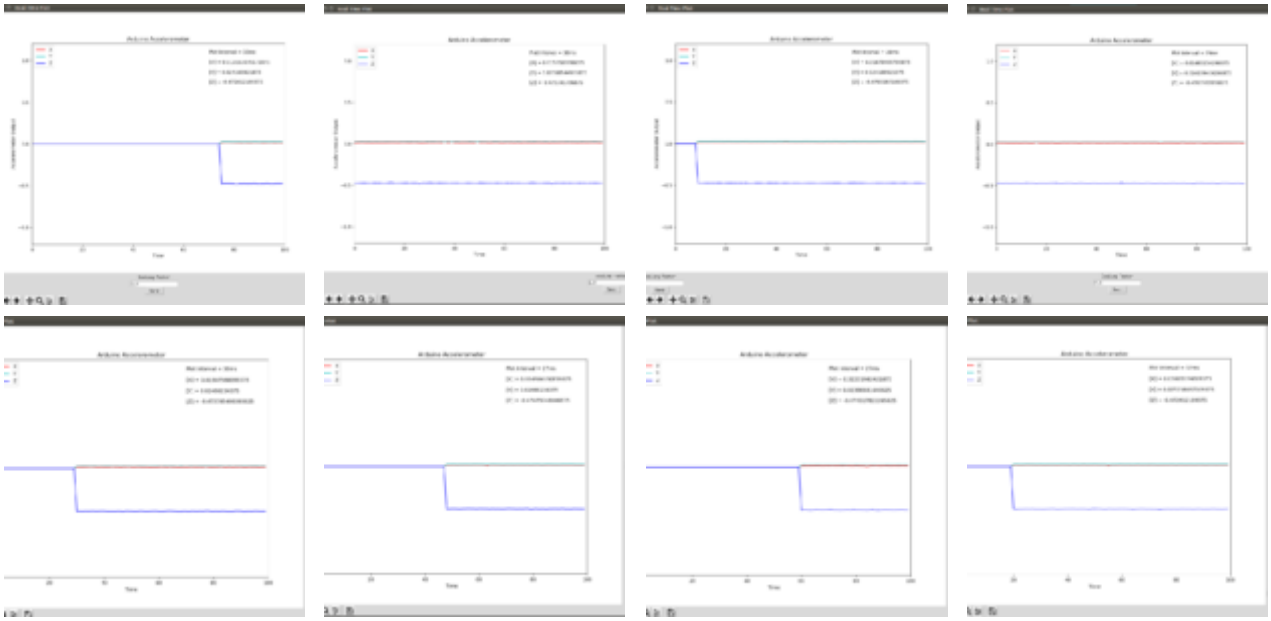


TKINTER PACK OBJECT PLACEMENT

Now, let us modify our Python code to incorporate the new frame for placing our widgets. We only have to modify the `initWindow()` method as follows:

```
def initWindow(self, figure):
    self.master.title("Real Time Plot")
    canvas = FigureCanvasTkAgg(figure, master=self.master)
    toolbar = NavigationToolbar2TkAgg(canvas, self.master)
    canvas.get_tk_widget().pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)

    # create a new frame to place our widgets
    frame2 = Frame(self.master)
    frame2.pack(side=Tk.TOP)

    lbl1 = Tk.Label(frame2, text="Scaling Factor")
    lbl1.pack(padx=5, pady=5)
    self.entry = Tk.Entry(frame2)
    self.entry.insert(0, '1.0')     # (index, string)
    self.entry.pack(padx=5)
    SendButton = Tk.Button(frame2, text='Send', command=self.sendFactorToMCU)
    SendButton.pack(padx=5)
```

In our case, Frame1 from the diagram above is actually the canvas (our graph). Thus, this is the case of both Frame1 and Frame2 being packed at the top. I have ran several combinations and the results are shown below.



Alright guys, this sums up the tutorial on creating a real time plot with GUI using Python. If you have any questions, please feel free to type it in the comments section below!

Go to Next Section

PREVIOUS POST
Plotting Serial Data from Arduino in Real Time with Python

NEXT POST
Motor Speed Control

POSTED IN FEEDBACK CONTROL SYSTEM

7 comments on "*Creating a Graphic User Interface (GUI) with Python*"

Fabian

August 4, 2018 at 12:15 pm

Hello, riky, first let me congratulate you on the project. I found it very interesting and I try to create something similar, I am working on a project based on the OPC UA protocol and I want to graph my data obtained by my client opcua. I was reviewing your code in python with matplotlib to be able to graph the data of my plant, but I have not yet been successful, the code is the following:

Actually, the code is not very complicated and I would like to know how the temperature data I obtain with my client can be included in a graphic animation in real time

Saludos de Ecuador

Fabian

August 4, 2018 at 12:16 pm

Hello, riky, first let me congratulate you on the project. I found it very interesting and I try to create something similar, I am working on a project based on the OPC UA protocol and I want to graph my data obtained by my client opcua. I was reviewing your code in python with matplotlib to be able to graph the data of my plant, but I have not yet been successful, the code is the following:

```
from opcua import Client
import time
url = "opc.tcp://169.254.81.206:4840"

if __name__ == "__main__":

client = Client(url)

client.connect()
print("Conexion Exitosa")

while True:
Temp = client.get_node("ns=1;s=Sensor.Nivel")
Temperatura = Temp.get_value()
print(Temperatura)

time.sleep(0.1)

client.disconnect()
```

Actually, the code is not very complicated and I would like to know how the temperature data I obtain with my client can be included in a graphic animation in real time

Saludos de Ecuador

rikisenia.L

August 4, 2018 at 3:45 pm

Hi Fabian,

This section talks about making a GUI which I don't think you need at the moment.
If you only want to show a real time graph, you can take a look at the previous section (https://thepoorengineer.com/en/arduino-python-plot/#python) instead.
You will, however, need to make some changes to the code that I have provided since you are not using a background thread to collect the data.
You should specifically read up on the animation.FuncAnimation() method (https://matplotlib.org/api/_as_gen/matplotlib.animation.FuncAnimation.html) to plot the graph in real time.

Adrian De La Cruz

October 23, 2018 at 11:41 am

Could you provide me with the Arduino library that you used?
I'm working on something similar and it would be useful to me.
Thank you! Really good job
Greetings from Mexico.

rikisenia.L

October 23, 2018 at 8:38 pm

Hi Adrian,

May I know which library are you referring to?
If you meant , it is available on my other post in this tutorial series.
Here's the link if you need it.
https://thepoorengineer.com/en/angle-control-absolute/

Gustavo

March 15, 2019 at 3:35 am

Thank you!
I'm new in Python, its code Works very well, Im tryng tom make an application where I need to make two graphics, and each of them has 3 plots, like the second example, but with two equals graphics, and I have not managed to do it, would appreciate very much you help, grettings!

luqman naseer

May 7, 2021 at 4:35 am

Hi.
I'm new on python can I try to use your accelerometer code but it show up some exception I make it right but when I compiled my program the graph window show nothing just a blank screen!! where should I sent the code for review?

Comments are closed.

Categories

Select Category

Archives

September 2019 (3)